# Disjunctive Scheduling with Setup Times: Optimizing a Food Factory

Nicolas Blais[1], Alexis Remartini[1], Claude-Guy Quimper[1], Nadia Lehoux[1], and Jonathan Gaudreault[1]

Université Laval, Québec, Canada
nicolas.blais.7@ulaval.ca, alexis.remartini.1@ulaval.ca,
claude-guy.quimper@ift.ulaval.ca,
nadia.lehoux@gmc.ulaval.ca,jonathan.gaudreault@ift.ulaval.ca

**Abstract.** In this research, we propose a new approach to solve scheduling problems applied to a food factory. The goal is to schedule the recipes in a way that minimizes the total setup time. Our model allows splitting a recipe in two batches if necessary. It also considers other specific constraints like avoiding too long setup during the day shift, having a maximum of tasks done during days shift, etc. A good heuristic based on the idea of scheduling the most difficult task as soon as possible is also presented. Moreover, we show how local search helps improving the solution.

**Keywords:** Preemptive scheduling · Food industry scheduling · Case study · Constraint programming · Local search.

## 1 Introduction

The food industry is an environment with many standards to respect that can quickly become complex if multiple products are made simultaneously on the same production line. In some factories, planners must manage the availability of the ingredients and of the production line in addition to the allergens of each product. They also have to take into account the setup time between all the products which depends largely on the allergens of the product. For example, the setup time typically takes much longer between a recipe that contains allergens and a recipe that does not contain allergens. This is mainly because workers must conduct an allergen clean-up for the entire production line between both recipes which can involve many hours.

The aim of this work is to use a constraint programming approach to schedule a set of tasks for a planning horizon over the weeks 5 to 8 with the objective of minimizing the sum of all setup times. The first 1 to 4 weeks are not considered (frozen horizon) because changes at this time generate many conflicts for the schedule and for the procurement of raw materials. The scheduling is recomputed weekly. The tasks must also be performed sequentially, i.e. we schedule on a single-machine.

In the literature, there are a few papers that use constraint programming applied to the food industry. Our model, therefore, contributes to the field by creating a set of constraints adapted for the heuristic to help the branching. Indeed, instead of branching directly on the starting time variable, we branch on the next variable and a set of constraints fixes the starting time variable. Also, we use local search strategies to improve the solution's quality for complex instances.

This paper is divided as follows. Section 2 first describes the problem. A literature review is proposed in Section 3. Section 4 and 5 describe the model developed and the heuristic used in the experimentation. The local search strategies tested are shown in Section 6 and we conclude in Section 7.

## 2 Problem Description

The problem under study in this research was observed in a food factory dedicated to cookies production. The company is a world recognized granola bars and cookies producer while offering a wide range of products. These products are, furthermore, offered without allergens. Planning such a complex products portfolio can be especially challenging. To help the planners in their work, we develop a model which could take their production system's dynamics into account.

We consider a set of tasks $T$ which correspond to cookie recipes. Each task $i \in T$ has a due date $LCT_{\mathrm{i}}$ called Latest Completion Times. In addition, a task $i$ must start after a date $EST_{\mathrm{i}}$ called Earliest Starting Time which is computed according to the availability and the preservability of the ingredients. Each task $i$ has a processing time $p_{\mathrm{i}}$ that is proportional to the number of cookies to produce. The transition time (or setup time) from a task $i$ to a task $j$ is given by $t_{i,j}$. All production lines are stopped on the weekend and some are also idle during the nights. These downtimes can be used to conduct a setup, but the production itself is stopped. So, for all working days $d \in D$, there is a time $shiftBegin_{\mathrm{d}}$ when the shift begins and another $shiftEnd_{\mathrm{d}}$ when the shift ends.

There are two kinds of tasks: *production orders* and *planned orders*. The production orders are already scheduled and cannot be moved. So, their starting times $S_i$ and production times $p_{\mathrm{i}}$ are known and fixed. For production orders, we therefore have $EST_{\mathrm{i}} + p_{\mathrm{i}} = LCT_{\mathrm{i}}$ which leaves no freedom on the time the task is scheduled. The planned orders needed to be scheduled so the time window in which the task must be scheduled is not tight: $EST_{\mathrm{i}} + p_{\mathrm{i}} < LCT_{\mathrm{i}}$. Moreover, these tasks can be executed in two segments in order to have a better use of the production line i.e. avoiding to have unused time before shift end. For example, a task can start on Monday, be stopped during the night and be restarted on Friday. At the end of its execution on Friday, the task must be completed, i.e., the time spent on Monday and Friday must sum up to the processing time. Nothing forces the two tasks to be adjacent in the schedule i.e. there may be other tasks running between them. When a task is executed in two parts, the duration of each of its part must be greater or equal to a threshold $minTime$. Tasks can

not be separated in more than two parts, because starting a new production can result in a drop in yield. Indeed, the first minutes of production are often unstable. So, separating a task into too many parts increases the chances of not getting the right amount of products in the allotted time.

There is a limit of $maxTaskDay$ tasks that can be achieved during a workday (or shift) to avoid that there are too many setups to make during a day. There is only one shift per day. Furthermore, only transition time whose duration is below a threshold denoted $maxSetup$ are permitted during a working day in order to avoid that the workers are idle for too long and to maximize the use of day time. In addition, there are other constraints that are not explained in this work because of their specificity and the brevity of this paper.

## 3 Literature Review

In the literature, Job-Shop Scheduling Problem (JSP) are typically solved using Mixed Integer Programming (MIP) methods and constraint programming (CP). The comparison between the two approaches in [5] showed that CP beats MIP for larger instances. It is mainly explained by the disjunctive constraint presented in [3] that is able to handle the setup time to make better filtering. As a result, using CP for our case becomes a natural choice.

The large neighbourhood search can, moreover, be used to improve the solutions quality, as demonstrated in [7]. They suggested starting the local search with simple strategies and making them more complex until the results are good enough because no strategy overtakes the others.

The strategies are based on those presented in [4]. Indeed, they proposed an approach that consists of first setting the precedence of all tasks, except some chosen randomly. Then, the solver tries to find better solutions.

## 4 Model

The model needed to help planners by scheduling a set of tasks for a planning horizon with a minimum of setup time. A constraint programming approach is used.

The model being preemptive, there are two parts for each task $i_1$ and $i_2$ that are the first and the second part of a task $i$ respectively. Let $S_{i_1}$ be the starting time of the first part of the task i and $\mathrm{dom}(S_{i_1}) = [EST_\mathrm{i}, LCT_\mathrm{i} - p_\mathrm{i}]$ be its domain. We assume that $\mathrm{EST_i}$ is always the beginning of a workday and one day is long enough to do the task in one segment. $\mathrm{LCT_i}$ is always at the end of a workday. We define $S_{i_2}$ be the same for the second part with the following domain: $\mathrm{dom}(S_{i_2}) = [EST_\mathrm{i} + p_\mathrm{i}, LCT_\mathrm{i}]$. Their domains are not the same because if the first part $i_1$ starts at $LCT_\mathrm{i} - p_\mathrm{i}$, the ending time would be at $LCT_\mathrm{i}$ (entirely done the task $i$). So, the second part $i_2$ would have a processing time of 0 and starts directly after the first part at $LCT_\mathrm{i}$. In what follows, when we use $T$, this includes the first and the second part of a task. Moreover, we add two dummy tasks (sentinel-begin and sentinel-end) at the beginning and at the end of the

schedule respectively. Their processing times are equal to 0 and they have no transition time from and toward them. They are also included in $T$.

The *next variable* $N_i$ is the task that follows $i$ in the schedule. We have $\text{dom}(N_i) = T \setminus \{i\}$. The objective function is to minimize the total setup time ($W$) of the scheduling for a production line, as shown in equation (1). For that, we declare a variable that represents the transition time of tasks $i$ toward its next task ($T_i = t[i][N_i]$). Also, we set the domain of $T_i$ to be the interval between the smallest and the largest transition time from a task $i$.

$$\text{Minimize: } W = \sum_{i \in T} T_i \tag{1}$$

The disjunctive constraint (2) introduced in [2] prevents two tasks from being executed simultaneously. It is used to filter the domain of $S$. This constraint takes as arguments two parameters. The first is the set of variable starting time $S$ and the second is the duration of a task $P_{\text{i}}$. The first variable is already defined with $S_i$, but for the second, we add the variable $P_{\text{i}}$ that represents the processing time of a task with the setup time associated to it ($P_{\text{i}} = p_{\text{i}} + T_i$).

$$\textsc{Disjunctive}(S, P) \tag{2}$$

To further improve the performance of the model by strengthening the filtering, we use a constraint called Weighted-Circuit (3) introduced in [1] representing our problem in the form of the Travelling Salesman Problem (TSP) with $t$ as distance matrix. For that, we force $N_{sentinel\text{-}end} = sentinel\text{-}begin$ to be able to form a circuit. $W$ representing the objective function.

$$\textsc{Weighted-Circuit}(N, t, W) \tag{3}$$

To model the fact that the production line is open between shiftBegin and shiftEnd, we use the constraints (4) and (5). For that, we use the variable $J_i$ which represents the working day when the task $i$ is executed ($\text{dom}(J_i) = D$). So, when a day $d$ is selected, the task $i$ is bounded between the $shiftBegin_{\text{d}}$ and the $shiftEnd_{\text{d}}$. Another advantage is that when a day $d$ is removed from the domain of $J_i$, the domain of the starting time of the task $i$ $S_i$ is then automatically filtered.

$$J_i \geq d \Rightarrow S_i \geq shiftBegin_{\text{d}}, \qquad \forall i \in T, \forall d \in D \tag{4}$$

$$J_i \leq d \Rightarrow S_i \leq shiftEnd_{\text{d}}, \qquad \forall i \in T, \forall d \in D \tag{5}$$

What differentiates this case with other scheduling problems is that a task can be done in two parts. As a result, to make the right amount of a product, we use equation (6). Also, the second part $i_2$ must imperatively be done after the first part $i_1$, as shown with constraint (7) so as to break symmetries. Constraint (8) expresses that the first part cannot follow the second part of the task to help the filtering.

$$\tilde{p}_{i_1} + \tilde{p}_{i_2} = p_{\text{i}}, \qquad \forall i \in T \tag{6}$$

$$S_{i_1} + \tilde{p}_{i_1} \leq S_{i_2}, \qquad \forall i \in T \tag{7}$$

$$N_{i_2} \neq i_1, \qquad \forall i \in T \tag{8}$$

To manage the fact that a task $i$ is executed in one single part (the first one), we declare the constraints (9) and (10) below. We assume that the transition time between two tasks with the same product (or the first and the second part of a task) is 0. So, if the first part of task $i$ produces the whole order, the second part has a processing time of 0 and must be done immediately after the first part. To assure that this task is executed on the same day, the constraint (10) is added.

$$\tilde{p}_{i_1} = p_{\mathrm{i}} \implies N_{i_1} = i_2, \qquad \forall i \in T \tag{9}$$

$$\tilde{p}_{i_1} = p_{\mathrm{i}} \implies J_{i_1} = J_{i_2}, \qquad \forall i \in T \tag{10}$$

In order to reduce the search space and obtain shorter computation times, we only allow the last task in a working day to be preempted. If a task $i$ cannot be fully executed in one single part, then its first part $i_1$ must finish at the end of a day shift ($shiftEnd$), as shown in constraint (11). We force $\tilde{p}_{i_1}$ to take the right value with the constraint (12). This constraint is redundant, but it helps filter the variable $\tilde{p}_{i_1}$. It is noted that the domain of $\tilde{p}_{i_1}$ is equal to $p_{\mathrm{i}}$ when $p_{\mathrm{i}}$ is smaller than 2 times minTime ($\mathrm{dom}(\tilde{p}_{i_1}) = p_{\mathrm{i}}$). Otherwise, the domain is $\mathrm{dom}(\tilde{p}_{i_1}) = [minTime, p_{\mathrm{i}}]$. Constraint (13) ensures that the minimum duration of a task ($minTime$) is respected. To achieve that, we make sure that if the first part does not entirely execute the task, then the time left is greater than $minTime$.

$$\tilde{p}_{i_1} < p_{\mathrm{i}} \implies S_{i_1} + \tilde{p}_{i_1} \in shiftEnd, \qquad \forall i \in T \tag{11}$$

$$\tilde{p}_{i_1} = \min(p_i, shiftEnd[J_i] - S_i), \qquad \forall i \in T \tag{12}$$

$$\begin{aligned} S_{i_1} + p_i > & shiftEnd_{\mathrm{d}} \implies \\ & S_{i_1} + p_i \geq shiftEnd_{\mathrm{d}} + minTime, \end{aligned} \qquad \forall i \in T, \forall d \in D \tag{13}$$

A specificity of the production lines concerns the maximum number of tasks that can be achieved during a day ($maxTaskDay$). Tasks with a null processing time are not included in the total number of tasks as they are not "real" tasks. We add a new variable $J'_i$ that is equal to the working day if the processing time is greater than 0 and equals -1 otherwise (see constraint (14)). Constraint (15) is a global cardinality constraint introduced in [6] that prevents the occurrences of each value of $J'_i$ from exceeding the maximum number of tasks allowed maxTaskDay.

$$\textsc{IfThenElse}(p_{\mathrm{i}} > 0, J'_i = J_i, J'_i = -1), \qquad \forall i \in T \tag{14}$$

$$\textsc{GlobalCardinalityLowUp}(J', maxTaskDay) \tag{15}$$

Large setups during a working day are not permitted. The duration of such a setup is bounded by the parameter $maxSetup$. The constraint (16) ensures that a transition beyond that threshold leads to a task that is performed another day. In other words, the setup time can be done during the night shift.

$$T_{i,j} > maxSetup \wedge J_i = J_j \implies N_i \neq j, \qquad \forall i, j \in T \tag{16}$$

## 5 Search Heuristic

In order to solve the model, we create a specialized search heuristic that branches on the variables next $N$, but looks at the domain of the starting time variable $S$ and the array of transition times $t_{i,j}$ to make the decision. Branching on the variables $N$ is a strategic choice. Indeed, if branching on $N_i = v_1$ leads to a failure, the solver will branch on $N_i = v_2$ which is a significantly different solution. If, however, the heuristic was branching on the starting time variables, say $S_i = 1$, upon a failure, the solver would try another value such as $S_i = 2$. However, starting the task $i$ one minute later is not a significantly different solution.

At the beginning, our heuristic chooses the dummy task sentinel-begin and makes it the current task. Let $i$ be the current task. The heuristic finds which value $v$ should be assigned to $N_i$. After the branching $N_i = v$ is performed, $v$ becomes the new current task. To select the value $v$, the heuristic chooses the task with the earliest starting time $S$ in its domain in the first place and breaks ties by selecting the task that is more difficult to schedule later. This is conducted based on these rules:

- Step 1: Choose the tasks $v$ that contains the smallest value in the domain $\text{dom}(S_v)$. If $v$ is unique, choose $v$, else go to Step 2.
- Step 2: For each task $v$, computed in Step 1, calculate the sum of all the transition times of a task not yet scheduled toward $v$. Go to Step 3.
- Step 3: Randomly choose the next task with probability proportional to the sum computed in Step 2.

We still need to assign a value to the starting time variables. Instead of branching on these variables, we add constraints to the model that force the starting time variables to take a value as the next variables get assigned through branching. They are not presented in this paper for lack of space.

## 6 Large Neighbourhood Search

Since the solver did not find an optimal solution (not even a good solution) in a reasonable time, we used a large neighbourhood search to improve it. This idea is still in development. We tested a few strategies. The methodology used is to first explore simple strategies and to then continue with more complex strategies, as suggested by [7].

The main idea of the local search used is to let the solver find the best solution in a given amount of time. After, we use a large neighbourhood search to improve this initial solution. For each iteration of the local search, the upper bound used for the objective is the best solution up to now minus 1. With that, we make sure that we have the best filtering with the weighted-circuit constraint.

The local search starts with an initial solution denoted as *best solution*. From this best solution, we partition the tasks into two sets: the free tasks and the fixed tasks. The fixed tasks are constrained to start exactly at the time they start in

the best solution. The free tasks can be assigned to any time point, as long as the constraints of the model are satisfied. An iteration consists of partitioning the tasks and solving the instance of the problem. This partitioned instance is easier to solve as no decision is required for the fixed tasks. If the solver finds a solution to the problem (a solution might not exist), this new solution becomes the new best solution. Then, we can repeat the process with another set of fixed variables until the termination criterion is reached. This can be done more than once since over the executions, the upper bound of the objective function is better, which leads to better filtering with the weighted-circuit constraint. In addition, the initial solution changes over the executions, which can lead to better solutions with large neighborhood search.

A way of doing this is to start with a solution and free all the tasks in the window for the first windowSize days and fix the other tasks. If a better solution is found, the *best solution* is updated. Then, we move forward the window by one day, free the tasks within that window and fix the tasks outside that window. The same process is repeated until the planning horizon has been covered. Figure 1 below shows this local search strategy. This can be repeated to achieve better results.

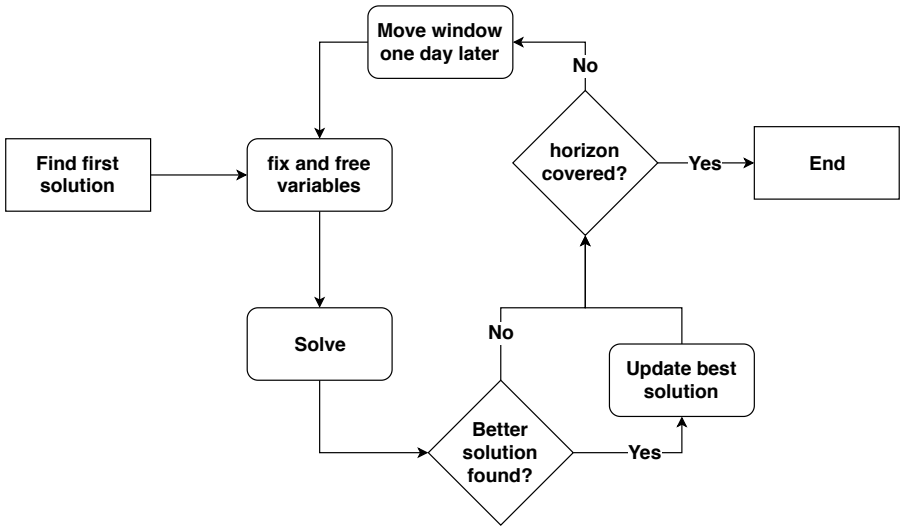


**Fig. 1.** Our Local Search Strategy

Moreover, another approach that can be tested is to free a window of a fixed number of consecutive tasks instead of consecutive days. As well as to make sure that the next variable $N$ is fixed rather than the starting time $S$ variable. In this way, it gives more flexibility for the local search and the process is more stable because there is always the same number of tasks relaxed in the schedule.

The next steps would be to use different branching heuristics for the local search. We hope to find out the one that would perform better for smaller problems. Also, we want to parallelize the local search process in order to use many strategies simultaneously. For example, we could use many heuristics for the same window or search for many non-overlapping windows at the same time. Finally, another possibility is to make the first run of the sliding window on all possible windows and note all the windows where the optimal solution was not found or lead to a great improvement of the solution. This information would then be used to conduct a second run that would put a greater amount of time on these windows.

## 7 Conclusion

This work proposes a solution for a scheduling problem applied to the food industry. The goal of this paper is to use constraint programming to schedule a set of tasks on a planning horizon encompassing the upcoming 4 to 8 weeks while minimizing the setup times. The goal is achieved by the development of a model, the use of a heuristic, and a local search. Future works will consist in exploring further local search strategies in order to improve the solution's quality.

**Acknowledgments:** The authors would like to thank Vincent Gingras who laid the basis of the model in the early stage of this project.

## References

1. Benchimol, P., Hoeve, W.J.v., Régin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**(3), 205–233 (2012)
2. Carlier, J.: The one-machine sequencing problem. European Journal of Operational Research **11**(1), 42–47 (1982)
3. Dejemeppe, C., Van Cauwelaert, S., Schaus, P.: The unary resource with transition times. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming. pp. 89–104. Springer International Publishing, Cham (2015)
4. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: International Conference on Automated Planning and Scheduling. vol. 5, pp. 81–89 (2005)
5. Ku, W.Y., Christopher Beck, J.: Mixed integer programming models for job shop scheduling: A computational analysis. Computers & Operations Research **73** (2016)
6. Oplobedu, A., Marcovitch, J., Tourbier, Y.: Charme: Un langage industriel de programmation par contraintes, illustré par une application chez renault. In: Ninth International Workshop on Expert Systems and their Applications: General Conference. vol. 1, pp. 55–70 (1989)
7. Wu, Y., Weise, T., Chiong, R.: Local search for the traveling salesman problem: A comparative study. In: 2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC). pp. 213–220 (2015)